2009 IEEE International Conference on Web Services

A Web Service Architecture for Decentralised Identity- and Attribute-based Access Control

Regina N. Hebig †, Christoph Meinel *, Michael Menzel *, Ivonne Thomas * and Robert Warschofsky †
Hasso-Plattner-Institute for IT-Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam
*firstname.lastname@hpi.uni-potsdam.de
†firstname.lastname@student.hpi.uni-potsdam.de

***Abstract*—The loosely coupled nature of Service-oriented Architectures raises the question how information for access control can be managed in an efficient way. Several specifications for Web Services exist to describe security requirements and to facilitate a provision of identity information. However, the integration of different standards regarding the expression of identity information in policies, claims and assertions comes along with an increased complexity. In order to identify and address the problems occurring with the combined use of standards as XACML, SAML and WS-Trust, we designed and implemented an architecture for identity- and attribute-based access control in decentralised environments. Our implementation provides an automated generation of access control policies in a format called XACML, a way to communicate required user attributes as claims across different domains based on the standards WS-Trust and WS-Policy, and a consistent mapping of retrieved attribute assertions to the XACML attributes in the access control policy.**

I. Introduction

Service-oriented Architectures (SOA) allow a seamless communication between applications independent from the platform on which they run and even across domain boundaries; therefore, making them perfectly suitable for the integration of services provided by independent business partners. However, such a loose coupling also implies new, critical security questions, as service providers need to protect their confidential resources from unauthorized access.

Authorization describes the process of determining whether or not a subject has the rights to perform a task, such as accessing a service. Hereby, the access control decision usually depends on the result of the authentication of the user who requests access and the provision of additional user attributes. While traditional systems tend to manage their users themselves, such an approach does not scale in a service-oriented environment, in which services are composed and re-used in different contexts. The management of user data such as registration and authentication information for each service leads to an explosion of user accounts and expensive maintenance costs in order to keep this information up-to-date. Approaches for the controlled sharing of identity and attribute information across multiple security domains evolved, which facilitate the re-use of existing identity and attribute management systems. This way, the management of users is decoupled in the architecture from the places where this information is required. In SOA, identity providers are the architectural entities which are designated to manage identity and attribute information and to answer requests for user information. Having decoupled the identity and attribute management from the access control at the service provider, several steps are necessary to perform the authorization. First of all, the service's conditions for granting access need to be expressed and published by the service provider. Further, this information has to be retrieved by the service user and communicated to the identity provider. In turn, appropriate formats and exchange mechanisms are required to transfer the requested attributes from the identity provider to the service and to interpret the received information with regard to the access control decision.

Several Web Service specifications and standards exist to address these tasks: WS-Policy [1] and WS-SecurityPolicy [2] can be used to describe security requirements of a Web Service, XACML [3] specifies a format to express access control policies, WS-Trust [4] defines the protocol to request and transfer security tokens and SAML [5] provides a standard token format to describe authentication, authorization and attribute assertions about a user.

Although these specifications are designed to complement each other, many questions are raised when combining them to secure a Service-oriented Architecture. In particular with regard to access control, these specifications describe the handling of user information that consists of different attributes of a specific type. Since attributes can be expressed differently or grouped to complex types, the same user information can be expressed in different ways. This ambiguousness complicates a mapping between different specifications. In addition, the manual specification of XACML access control policies and the corresponding requirements described by claims in WS-SecurityPolicy is cumbersome and error-prone. The definition of XACML policies itself is difficult due to the complexity of this specification.

To address these challenges,

- we describe a prototype implementation with an architecture based on the standards XACML, SAML, WS-Policy, WS-SecurityPolicy and WS-Trust which puts the focus on sharing identity and attribute information across independent domains for the purpose of access control.

- we provide an easy, declarative way of stating required attributes, which are translated to XACML access control policies in an automated manner.
- we enable an automated translation of XACML to WS-Policy, which exposes the requirements regarding identity information as claims.
- we define a consistent mapping of retrieved attribute assertions expressed as SAML tokens to the XACML attributes in the access control policy in order make the policy enforcement independent of application-specific policies.

Following the introduction, Section II gives a short overview of used standards. Section III presents the architecture and goes into details about the implementation of the authorization process based on the architecture. Section V gives an overview about related work in this area. Finally, Section VI concludes this paper and highlights some future work.

II. Used Standards

Several tasks need to be accomplished in order to realize a scenario in which the management of information required for access control is decoupled from the place where the access control step is performed. First of all, an access control policy has to be defined and enforced at the service side. The eXtensible Access Control Markup Language (XACML) [3] is a general XML-based policy language, which is a well established standard to define and enforce policies. As part of the standard, the specification describes an authorization architecture. The core concept of this architecture is a policy decision point (PDP). The PDP decides about the authorization. A policy enforcement point (PEP) requests the PDP for a decision about an authorization, based on information extracted from the message and its context. Furthermore, the WS-Policy [1] specification can be used to communicate the required information for access control such as assertions about the identity to the service consumer. While XACML provides the means to express and enforce a policy, it does not specify how to request and retrieve the required credentials. Therefore, WS-Trust [4] has been specified as a standard token request protocol, which can deal with several token formats. One standard for such a security token format is the Security Assertion Markup Language (SAML) [5], which can be used to express assertions about a user's authentication, authorization and attributes.

III. A SAML/XACML based Solution for Decentralized Attribute-based Access Control

We designed and implemented an architecture for identity- and attribute-based authorization with the focus on an efficient handling of user attributes. In particular, we focused on the specification of service requirements concerning user attributes and the mapping of received attribute values from the security tokens to the attributes in the access control policy. Our solution comprises three steps, in which the authorization process is performed: the *policy definition*, the *credential retrieval* and the *policy enforcement*.

A. Authorization Prozess

The *policy definition* is the first step of an authorization process, which has to be done by the service owner. Defining a policy is a complex and tedious task, which bears significant security risks, if not done properly. Our approach is based on an automated policy generation to reduce the likelihood of errors. It preserve the consistency and uniform structure of the policy, which in turn enables a consistent mapping between SAML and XACML attributes during the enforcement of a policy. This aspect is explained in more details in the part *policy enforcement*. Another benefit of the policy generator is that we can associate attributes with unique URIs in an automatic manner. This allows the unique identification of attributes across the border of the service. The idea of using URIs for the identification is taken from the *Information Card Profile* [6], which associates some standard attributes to URIs within an XML schema file. An example for such an URI is *http://schemas.xmlsoap.org/ws/2005/05/-identity/claims/emailaddress* which expresses the email address of a user.

In addition to the generation of the policy, a description of the attributes is automatically generated as a kind of metadata which can be shared with other participants in the SOA infrastructure. The format is described by an XML schema.

Once the policy is defined, the second phase of the authorization process is the *credential retrieval*. In this phase, information about required attributes needs to be communicated to the service requester and to the identity provider (IP) who will process the request for an attribute. In order to understand requests for attributes which are not standardized, the identity provider can resolve the URI, which identifies the attribute in a global context. This way, the IP becomes independent from the service. Since an attribute is identified by a global, unique URI, the local occurrences of the attribute as for example in the IP storage or in the service domain can be matched even if they have different names.

The last authorization phase is the *policy enforcement*. During policy enforcement attributes extracted from the security token received with the SOAP message need to be mapped and matched with attributes in the XACML access control policy. This requires an attribute of the request to be located in the same group as in the policy. XACML distinguishes between four attribute groups: Subject, Object, Action and Environment. For an automated generation of the access request out of the security token received with the SOAP message, this mapping to the attribute group has to be unambiguous. To ensure this, we define restrictions with regard to the XACML attribute group an attribute can belong to. Since we are generating the XACML policy in an automatic manner, we can assure that these restrictions are followed. The benefit of this procedure is, that the generation of the XACML request does not need any information about the XACML policy itself anymore; therefore making this mapping independent from the current XACML policy.

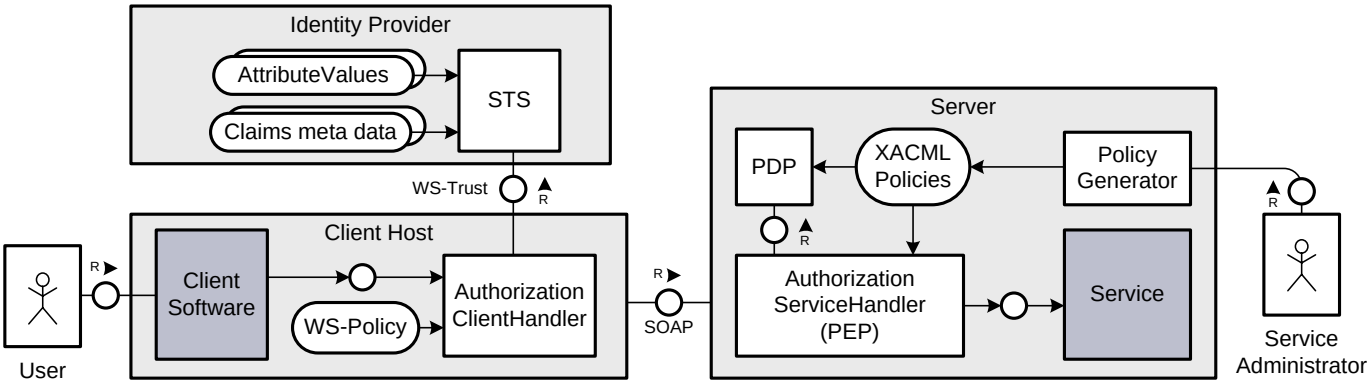

Fig. 1. FMC Block diagram [8] of the authorization handler architecture.

B. Architecture

Our architecture comprises the main entities of a SOA as specified by the Liberty Identity Web Service Framework (ID-WSF) [7]: a client, a service and an identity provider. The identity provider offers a Security Token Service to retrieve security tokens for the identity and attribute information it manages. A client in our architecture is a piece of software the subject uses to request a service.

Our implementation is based on the handler concept of the Web Service framework Axis2 [9] to allow an easy integration with other service implementations. During the authorization process, several handlers are involved. On the client side, a handler is required to add information about the user to the SOAP message. On the service side, a handler is needed to check the received information and to decide whether or not the subject of the transaction is authorized to use the requested service. A detailed overview of the architecture given in FMC notation [8] is shown in Figure 1.

The handler at the client side is located in the *out flow* chain of the client's Axis2 handler chain. Upon request, it inserts a SAML assertion received from the STS into the SOAP header of the outgoing message, including information like the subject and the issuer of the assertion as well as several attributes about the subject. In order to request this information from the Security Token Service via the WS-Trust protocol, the client needs to have access to the WS-Policy of the service. The service policy according to the WS-Policy specification is usually transmitted from the service to the client upon request. It contains a list of attributes that are needed for the authorization.

The handler on the service side is responsible for extracting the SAML assertion from the incoming message, creating a PDP (policy decision point) instance, configuring this instance with a given XACML policy and requesting access at the PDP with the information resolved from the message.

As shown in Figure 1 the service handler and the PDP both need the XACML policy. This XACML policy is created by a 'Policy Generator' based on inputs from the service administrator.

IV. Implementation

This sections gives an insight into our implementation. The emphasis is put on our improvements with regard to an efficient handling of identity and attribute information from multiple sources for the purpose of access control. Section IV-A will start with a description of our automated definition of access control policies, then section IV-B will describe the retrieval of credentials, and finally section IV-C will give details about the policy enforcement step.

A. An automated Policy Definition

In many cases, access to a Web Service is restricted to certain users identified by specific attribute values such as a specific role. These attributes can be of a very global nature like a name or the age of a person. However, as in many cases such terminologies are used within companies, the character and names of the attributes might be more specific or proprietary.

Thus, the definition of an XACML policy for access control is an individual and tedious task. The following list states several facts that make the definition of policies difficult.

Complexity: The first challenge arises from the complexity of an XACML policy. The more complex a policy gets, the harder it is to maintain without proper tool support; leading to inconsistencies and errors due to the human factor involved.

Expressiveness: A second problem with such policies is the vast expressiveness of the XACML standard. Most access restrictions can be stated in many different ways. Especially, the mapping of attributes to one of the four attribute groups in XACML can differ between policies, since different people sort the attributes in a different way. Although the groups have a useful semantic, it is not possible for a computer to guess the right group of an attribute automatically.

Attribute identity: As soon as an application needs more customized behaviour, it is likely that the required attributes are so specific that they are not covered by global definitions, such as provided by the XML schema from the InformationCard Profile: *http://schemas.xmlsoap.org/ws/2005/05/-identity/claims.xsd* [10].

Nevertheless, it is desirable to have a common understanding about an attribute's format and meaning and match

attributes in different domains which have the same meaning, but might have different identifiers as for example 'last name' and 'surname'.

Our architecture is designed to address these problems. By providing a policy generator, we hide the complexity of an XACML access control policy from the service administrator. Therefore, we restrict XACML policies to a structure which is sufficient for the purpose of attribute-based access control, but solves the problems due to the expressiveness of XACML.

In order to deal with the problem of attribute identity we use URIs to identify them. By using this kind of unique identification, it is possible for the STS to find the correct attribute, even if it has another name in the STS workspace than in the workspace of the service.

When defining an attribute with the policy generator, the generator automatically creates an adequate XML schema to describe the structure of the newly introduced attributes. This schema file can then be deposited under the URI of the attribute. To express the requirements of the service, the policy generator finally generates a description of required attributes, a so-called `<Claims>`-tag for the WS-Policy of the service. This `<Claims>`-tag includes the URIs of all required attributes.

In the following, we describe the restrictions we made for the XACML policy. After that we describe the architecture of the policy generator.

1) Extending the XACML profile for Web Services: Many rules for access control can be expressed in different ways. This leads to a high degree of complexity and makes the evaluation of the policy in the enforcement step more difficult. In order to formulate an access request, the application-specific attributes used in the policy need to be known. This makes it difficult to automate the handling of policies in an application-independent context. Anne Anderson also refers to this problem in [12].

In our implementation, we take the XACML profile for Web Services [11] as a foundation and add additional restrictions to the structure of an XACML policy. These restrictions are done in a way that still every rule for access control can be expressed, but the number of different ways it can be expressed is restricted. The XACML profile for Web Services describes how XACML can be used to protect Web Services from unauthorized access. Therefore, the profile restricts the possibilities of the XACML policy 1.1 standard to achieve a consistent way for expressing access constraints in XACML policies in the following way:

- The targets of the policy sets and of the policy itself are restricted to four variables: *objective id*, *port id*, *operation id* and *message id*. The *objective id* 'indicates the aspect of a policy addressed by a `<Policy>` element ...'[11]. It defines the target of the overall policy. Since we are dealing with policies for access control, the *objective id* is set to 'authorization'. The *port id* defines the target of the outermost policy set and equals the port id defined in the WSDL of the requested SOAP service. The *operation id* and the *message id* are also equal to the corresponding ids of the SOAP service. These ids are used to define the inner policy set.
- Further, the profile postulates that the effect of a rule is always 'permit'. This means no explicit deny can be defined. This way, we can later interpret automatically every response of the PDP as 'deny' if it is not explicit 'permit'.
- Furthermore, all parts of a condition shall be connected by a logical `and`. This means that all parts of the condition have to be fulfilled in order for the condition to evaluate to true.

In addition, we introduce additional restrictions.

- As aforementioned, the XACML specification defines four attribute sets. An XACML authorization request requires the attributes to be given in the right attribute set. Therefore, we restrict the XACML profile by predefining the mapping to the attribute set. First of all, a policy contains the subject attributes, which specify the subject refereed to in access control policy. In our approach, there is exactly one subject. This is the subject of the security token, the SAML assertion. Therefore, we define that this subject belongs to the URI `saml/subject/name` and belongs to the subject attribute set.
- We further define all other attributes to be part of the environment attribute set. This includes the name of the issuer (the STS who signed the security token) of the SAML assertion. Since the issuer, like the subject, is part of every SAML assertion we defined for it the URI `saml/issuer/name`.

Since we limit only the number of ways to express the same policies, the expressiveness stays the same as in the XACML Profile despite the defined restrictions.

2) The policy generator: Above we identified some problems, namely complexity, expressiveness and attribute identity. We also introduced the XACML profile and our additional restrictions, which solve the expressiveness problem.

However, a human administrator still has to perform a complex task. The administrator has to choose the right configuration for the target of the policy and single policy sets. He has to consider the XACML syntax for the expression of conditions. How to express that one attribute has to have a special value? Additionally, the wrapping of expressions in conditions and rules has an effect to the semantic. Are two expressions alternatives or have both of them to be fulfilled?

All this has to be done conform to the XACML profile and our restrictions, because otherwise an automated handling of the policy is impossible.

Obviously, this is a very complex and error-prone task. In order to solve this problem our architecture includes a policy generator. The current policy generator is designed to prove the feasibility of generating policies from a fixed set of infomation. It generates XACML Policy files for a given port id and given policy sets. The policy sets contain rules that define the requested attributes and attribute values. Listing 1 shows some example code which defines a rule `addMember`

which restricts access to the operation addMember to all users owning the role hpistaff.

```
PolicySet ps = new PolicySet();
List<Rule> rules = new ArrayList<Rule>();
Rule rule = new Rule("addMember");

rule.addAttributeAssignment(
    AttributeAssignment.newAttribute(
        "member_group",
        "hpi_staff",
        PolicyGenerator.DataTypes.string,
        PolicyGenerator.Function.equal));
rules.add(rule);

ps.addPolicy("addMemberPolicy", rules);
String[] operationids =
    new String[] { "addMember" };
ps.set_operationIds(operationids);
```

Listing 1. Example code for the generation of a policy set

As can be seen from Listing 1, the use of this interface is much less complex and error-prone than writing an XACML policy by hand. Using a graphical user interface based on our generator, the task of defining a XACML policy becomes manageable with a few clicks.

Simultaneously to the XACML policy file, also an XSD schema file is generated. This XSD file defines the elements for the claims of the attributes and their types. On the basis of this, the identity provider can select the attributes that are required by the service, even if the names of the attributes are not the same. This solves the attribute identity problem.

Furthermore, a WS-Policy file which contains the claims is also generated. This policy file can be merged with an existing policy file describing the other requirements of the service, so that a client has to use only one policy file to know which information has to be send to the service.

B. Policy Credential Retrival

In this section we present the mechanisms to resolve the attribute requirements from the policy into corresponding security tokens and describe the implementation of the involved architecture components, namely the client side handler and the STS. In order to request access to a service, the client side handler first has to retrieve and analyse the WS-Policy document and extract the claims from it. For each claim, a corresponding security token, which contains proof of the claim has to be requested from an STS via the WS-Trust protocol.

Upon request, the STS issues a security token containing an attribute statement. This attribute statement must contain the attributes, claimed in the WS-Policy. Therefore, the STS has to obtain the requested attribute values. Finally, the client side handler inserts this security token, which is a SAML assertion, into the security header of the service request.

The remainder of this section is structured as follows. First we introduce how the client side handler deals with the policy of the service and communicates with the STS. After that we introduce how the STS resolves claims into actual security tokens.

1) The client side handler: The client side handler has the task to extract the claims from the WS-Policy, to request a SAML assertion at the STS and to insert this assertion into the message header of the service request.

Inside WS-Policy documents, claims are used to declare, which information is needed by a service. Since WS-Policy documents are XML files, the claims are encapsulated in a <Claims> tag that declares the dialect. The dialect identifies a group of claims, which are defined at the same location (e.g. the same XML schema file). Each required attribute is then declared in a <ClaimType> tag by an URI. The definition of a dialect is done in an XML schema file.

Following the WS-Trust standard, the client side handler has to extract this <Claims> tag from the WS-Policy. After this, it adds this tag to a request security token (RST) message as child of the <RequestSecurityToken> tag. This RST message is sent to the STS within the body of a SOAP message.

Figure 2 shows an example of a RST message including a <Claims> tag. The <TokenType> tag defines what kind of token is requested. As we choose SAML assertions for the communication, we can see in this example that a SAML assertion is requested. The <RequestType> tag defines the kind of request. Finally, the <Claims> tag defines attributes which have to be included in the assertion nested in an <AttributeStatement> tag.

```
<wst:RequestSecurityToken
    xmlns:wst=".../ws-trust/200512">
  <wst:TokenType>
    http://.../oasis-wss-saml-token-profile-1.1#SAMLV2.0
  </wst:TokenType>
  <wst:RequestType>
    http://.../ws-trust/200512/BatchIssue
  </wst:RequestType>
  <wst:Claims
    wst:Dialect="http://myPort/claims"
    xmlns:ic="http://schemas.xmlsoap.org/ws/2005/05/identity">
    <ic:ClaimType
      URI="http://myPort/claims/myattribute"/>
  </wst:Claims>
</wst:RequestSecurityToken>
```

Fig. 2. Example of a Request Security Token according to WS-Trust [4]

The STS has to trust the client in order to accept the user's authentication information it receives within the SOAP message from the client. As the mechanisms of authentication are well standardized and probed, it is out of scope of this paper to describe this aspect in detail.

The STS returns the requested SAML assertion within a request security token response (RSTR) message. Finally the client extracts the assertion (the <saml:Assertion> tag) and inserts it into the header of the service request.

2) The STS module: If the STS gets a request for a security token that can be authenticated, it resolves the required attributes in order to return an appropriate RSTR message. Therefore, the STS iterates through the attributes in the <Claims> tag of the RST message.

For each <Claims> tag the STS consults the metadata of the attributes. In a simple implementation the STS would hold XML schema files (XSD), which define attribute URIs, for every supported service. However, to follow up the vision of a flexible SOA, the STS can use the URI of each attribute to resolve the appropriate metadata information over the web; therefore making the STS independent from the services.

After consulting the metadata, the name of the attribute and its type (in our implementation this can be integer or string) is known. With this information the requested data can be extracted from the storage, which contains the attribute values and can be inserted in an <AttributeStatement> element. Later on, this <AttributeStatement> element is included in the SAML assertion as the third child beneath the <Issuer> tag, which refers to the STS and the <Subject> tag, which identifies the user.

The access to the attribute data is done through the interface IAttributeResolver. This allows changing the implementation of the access and with it the type of storage. If for example the data should be stored in a database, the location and structure of the attribute values can easily be changed by substituting the implementation of this interface.

Figure 3 shows an example of a RSTR message. The requested SAML assertion is included in the <RequestedSecurityToken> tag. As said above the assertion includes an <Issuer> tag and a <Subject> tag for the information about issuer and subject. The claimed attribute data is included in the <AttributeStatement> tag.

```
<wst:RequestedSecurityTokenResponse
    xmlns:wst="http://.../ws-trust/200512">
  <wst:TokenType>
    http://.../oasis-wss-saml-token-profile-1.1#SAMLV2.0
  </wst:TokenType>
  <wst:RequestedSecurityToken>
    <saml:Assertion
        xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
        Version="2.0">
      <saml:Issuer>myIssuer</saml:Issuer>
      <saml:Subject>
        <saml:NameID>mySubject</saml:NameID>
      </saml:Subject>
      <saml:AttributeStatement>
        <saml:Attribute Name="myattribute">
          <saml:AttributeValue ...>
            ...
          </saml:AttributeValue>
        </saml:Attribute>
      </saml:AttributeStatement>
    </saml:Assertion>
  </wst:RequestedSecurityToken>
</wst:RequestedSecurityTokenResponse>
```

Fig. 3. Example of a Request Security Token Response according to WS-Trust [4]

C. Policy Enforcement

The last part of the authorization process, handled by our architecture is the *policy enforcement*. During the policy enforcement the access request is evaluated against the policies of the service.

After receiving a request, the service has to check its own policies and to decide whether or not the requester has permission to access the service. Therefore, the service has to inspect the SOAP message and extract the token, which is the input for the access control decision. As we use SAML this token is a SAML assertion located in the header of the SOAP message.

The important information for access control in a SAML assertion are the subject and the issuer of the assertion as well as all statements about the subject. This information has to be converted into the standard request format for the policy decision point (PDP). As we use XACML for access control the service must map the attributes defined with the SAML language to XACML attribute definitions to request the PDP.

1) The mapping from SAML to XACML: To be specific, attributes in the SAML assertion need to be mapped to policy sets of XACML. This mapping is listed in table I.

Analogous to the previously made restrictions we simply map the subject attribute of the SAML assertion to the attribute with the URI saml/subject/name, which belongs to the subject attribute set.

The resource attribute set shall contain attributes describing the requested resource. Therefore, according to the XACML profile this set contains the port id and the message id. The operation id does not specify the requested resource but the action to be performed on the resource. This means the operation id is an attribute of the action attributes set like the objective id 'authorization'.

Again, according to the restrictions we defined all other possibly important attributes are part of the environment attributes set. The issuer of the SAML assertion is put in this attribute set together with all attributes contained in the SAML assertion. To match with the attributes specified in the XACML policy the names of the SAML assertion attributes must equal the corresponding URIs in the XACML policy and the types must also be the same.

SAML Attributes	XACML Attribute Set
SAML Subject	Subject Attribute Set
SAML Issuer SAML Assertion Attributes	Environment Attribute Set
WSDL Port ID WSDL Message ID	Resource Attribute Set
WSDL Operation ID	Action Attribute Set

TABLE I
MAPPING SAML ATTRIBUTES TO XACML ATTRIBUTE SETS

2) Requesting the Policy Decision Point: The PDP we use is part of the XACML library from Sun [14]. In order to query it, we create an instance of the PDP and send an XACML request. The request is constructed with the four attribute sets described above. It can be serialized to XML. An example request is shown in Figure 4.

The PDP evaluates this request with the XACML policies it has access to. After that, the PDP delivers a response, such as shown in Figure 5.

```
<Request>
  <Subject SubjectCategory="...">
    <Attribute AttributeId="saml/subject/name"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="...:portId"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
    <Attribute AttributeId="...:messageId"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
    ...
  </Resource>
  <Action>
    <Attribute AttributeId="...:objectiveId"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>authorization</AttributeValue>
    </Attribute>
    <Attribute AttributeId="...:operationId"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
  </Action>
  <Environment>
    <Attribute AttributeId="saml/issuer/name"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
    <Attribute AttributeId="..."
        DataType="http://www.w3.org/2001/XMLSchema#...">
      <AttributeValue> ... </AttributeValue>
    </Attribute>
    ...
  </Environment>
</Request>
```

Fig. 4. Example of a XACML request

```
<Response>
  <Result ResourceID="/ID">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="...:status:ok"/>
    </Status>
  </Result>
</Response>
```

Fig. 5. Example of an XACML response.

There are four possible responses: *permit*, *deny*, *not applicable* and *indeterminate*. In the case of *permit* the access to the requested SOAP service is granted, otherwise not.

V. Related Work

Various major authorisation systems exist that are traditionally based on a centralised approach to be deployed in a single trust domain. With the advent of Web Services and Service-oriented Architectures, these frameworks are enhanced to meet the requirements of a distributed environment. For example, the Privilege and Role Management Infrastructure Standards Validation (PERMIS) framework is a policy-based authorization infrastructure implementing a hierarchical Role Based Access Control (RBAC) model [15]. It has been designed to be used in a single trust-domain and is based on a privilege management infrastructure to maintain the user attributes in X.509 attribute certificates (AC), which are published in LDAP directories. An extension is described in [15] to enable PERMIS to issue SAML authorization tokens in order to use PERMIS as a central authorization service in a distributed environment. Since the negotiation of requirements for access control is not supported, the usage of this approach in an environment based on loosely coupled services is restricted.

Another approach is the proposed multipolicy authorization framework for Grid infrastructures by Lang *et al.* in [16]. This framework is based on the specifications XACML and SAML. The authorization mechanisms of the Grid computing platform support multiple security policies and dynamic policy changes. A PEP intercepts a user's access request and executes the authorization decision of the PDP. To decide whether an authorization is granted or not, the requested PDP chooses a dedicated PDP for each type of policy that is used in the Grid architecture. The access decisions are based on a requester's attributes, such as the service, the resource, or the environment. The general concept of this framework describes a centralized resource and authorization management that does not fit into the decentralised concept of Service-oriented Architectures. Although this concept describes a promising approach to enforce different types of policies, the translation and communication of security requirements in a distributed environment is not in the scope of this work.

Cardea [17] as another distributed authorization framework protects potentially accessed resources through local access control policies that are specified within the XACML syntax. It dynamically evaluates authorization requests according to a set of relevant characteristics of the request and the requester. The system tries to abstract from locally defined identities and, therefore reduces the amount of user information for access control.

In general, these frameworks assume a tight binding between the identity provider and the service. Therefore, an intense negotiation and exchange of user attributes between both parties is not required, making them less suitable for the vision of SOA to provide a global market place of ubiquitously available services. With regard to the combination of standards, several profiles have been developed which describe how one standard can be used in the context of another one. As one of these profiles which is widely accepted and ratified as an OASIS standard, the SAML profile of XACML describes the integration of SAML and XACML. It was proposed by Anderson and Lockhart [18] in 2005.

Many solutions for authentication and access management exist which use these two specification SAML and XACML in combination for different purposes. Schlaeger et al. [19] provide an holistic approach to attribute-based access control in Service-oriented Architectures. Their solution uses XACML and SAML and is based on the architecture of the Liberty Alliance Project featuring an identity provider and several service providers. The focus of their prototype clearly lies on preserving the privacy demands of the user; therefore including a privacy-enhancing protocol into their solution. They do not consider metadata for attributes nor the policy definition

process. Moreover, their implementation is based on SAML 1.1 while we support SAML 2.0.

VI. Conclusion

To grant the right people access to the right information at the right time is crucial to improve efficiency of business collaborations. Therefore, a strong and secure authentication and authorization infrastructure is a must, which can adapt flexibly and quickly to fast changing business and security requirements.

In this paper, we describe our implementation of an architecture to support the authorization in a decentralized environment, in which identity and attribute information can be managed by independent entities. Our architecture comprises the main elements for decentralized authorization as described by the Liberty Identity Web Service Framework [7]: identity provider, service providers and client. We combine the standard protocols and formats of XACML, SAML, WS-Policy and WS-Trust to perform a decentralised user authorization, which consists of the steps: policy definition, credential retrieval and policy enforcement. While the general authorization process is well understood, our focus lies on improving the authorization process in specific aspects. First of all, we ease the policy definition process by providing a policy generator for XACML and WS-Policy. Our policy generator takes as input solely a set of required user attributes declared by type, name and value and generates corresponding XACML and WS-Policy documents. This frees a service administrator from understanding the complicated meaning of policies and from working directly with XML elements. In addition, consistency between the different policy formats XACML and WS-Policy is ensured.Moreover, our implementation addresses the problem of describing the user attributes which are required for the access control decision. As described by the InformationCardProfile [6], we associate user attributes to a dialect which points to an URI, which provides meta information for user attributes within a policy. This allows for a looser coupling between identity and service provider, which is, in particular, an important pre-requisite to provide efficient communication across multiple domains.

We believe that these two aspects – metadata for attribute policies and a more intuitive policy definition – are important aspects in any authorization architecture. Since our implementation is purely based on open standards, our results can be used for integration into other authorization processes. Moreover, our implementation is based on the handler architecture of Axis2 and therefore flexible enough to allow an easy integration with other service implementations.

In our architecture, the attributes that are the basis for the access control decisions, are defined in XSD schema files. In future work, we intend to extend this definition with a semantic meaning by using a language like OWL or RDF instead of XSD schema files. Having semantically defined attributes, a mapping between two attribute which are syntactically different but semantically equal, can be done easily. Especially complex attributes would profit from this.

Furthermore, with regard to automated service compositions, a correct merge of multiple policies which contain semantically related attributes would be possible.

References

[1] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. von Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, mit Yalinalp, and D. Orchard, "Web Services Policy 1.2," W3C, http://www.w3.org/Submission/WS-Policy/, Tech. Rep., apr 2006.

[2] G. Della-Libera, M. Gudgin, and et all, "Web Services Security Policy Language (WS-SecurityPolicy)," Public Draft Specification, Juli 2005. [Online]. Available: "ftp://www6.software.ibm.com/software/developer/library/ws-secpol.pdf"

[3] T. Moses, "eXtensible Access Control Markup Language (XACML) Version 2.0," OASIS, Tech. Rep., 2005.

[4] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, "WS-Trust 1.3," OASIS Standard Specification, Organization for the Advancement of Structured Information Standards (OASIS), 2007, OASIS Standard.

[5] S. Cantor, J. Kemp, E. Maler, and R. Philpott, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.02," OASIS Standard Specification, 2005. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/

[6] A. Nanda, "A Technical Reference for the Information Card Profile V1.0," http://msdn.microsoft.com/en-us/library/bb298802.aspx, Microsoft Corporation, Tech. Rep., Dec. 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb298802.aspx

[7] J. Tourzan, Y. Koga, M. Aoyagi, R. Aarts, J. Beatty, C. Canales-Valenzuela, G. Ellison, J. Hodges, J. Kainulainen, J. Kemp, P. Madsen, J. Rouault, P. Thompson, and T. Wason, "Liberty ID-WSF Web Services Framework Overview, Version: 2.0," Liberty Alliance, 2006, non-normative specification.

[8] A. Knpfel, B. Grne, and P. Tabeling, *Fundamental Modeling Concepts*. John Wiley & Sons Ltd, 2005.

[9] Apache, "Apache Axis2 Version 1.4," Apache Software Foundation, http://ws.apache.org/axis2/, Tech. Rep., 2008.

[10] D. Chappel, "Understanding Windows CardSpace," April 2006. [Online]. Available: http://msdn2.microsoft.com/en-gb/library/aa480189.aspx

[11] T. Moses, A. Anderson, S. Proctor, and S. Godik, "XACML profile for Web-services," OASIS, Tech. Rep., 2003.

[12] A. H. Anderson, "Domain-independent, composable web services policy assertions," in *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 149–152.

[13] S. Olshansky and S. Cantor, "OpenSAML," https://spaces.internet2.edu/display/OpenSAML, Jun 2008. [Online]. Available: https://spaces.internet2.edu/display/OpenSAML

[14] Sun Microsystems Inc., "Sun's XACML Implementation," Sun Microsystems Laboratories, http://sunxacml.sourceforge.net, Tech. Rep., 2006.

[15] D. Chadwick, S. Otenko, and V. Welch, "Using SAML to Link the GLOBUS Toolkit to the PERMIS Authorisation Infrastructure," in *Proceedings of Eighth Annual IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, Windermere, UK, September 2004.

[16] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A multipolicy authorization framework for grid security," in *NCA '06: Proceedings of the Fifth IEEE International Symposium on Network Computing and Applications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 269–272.

[17] R. Lepro, "Cardea: Dynamic Access Control in Distributed Systems," *Technical Report TR NAS-03-020, NASA Advanced Supercomputing Division*, 2003.

[18] A. Anderson and H. Lockhart, "SAML 2.0 profile of XACML v2.0," OASIS Standard Specification, February 2005.

[19] C. Schläger, T. Priebe, M. Liewald, and G. Pernul, "Enabling attribute-based access control in authentication and authorisation infrastructures," *Proc. of the 20th Bled eConference-eMergence (Bled 2007)*.